A Digital/Analog Signal Processing Method to Compute the Number of Hamiltonian Paths

Bryce Kim

01/07/2017

Abstract

This paper explores finding the number n_h of undirected hamiltonian paths in an undirected graph $G = (V, E)$ using lumped/ideal circuits, specifically low-pass filters. Ideal analog computation allows one to computer n_h in a short period of time, but in practice, precision problems disturb this ideal nature. A digital/algorithmic approach is proposed, and then it is shown that the approach/method operates under theoretically feasible (polynomial) time: $O(n^{230}) + O(n_d PC(p_2, n_d))$, where $n = |V|$ and $PC(p_2, n_d)$ refers to the time complexity of finding a root of a polynomial with polynomial coefficients being p_2 digits (with coefficients limited within $2^{p_2/2}$ and $1/2^{p_2/2}$ in magnitude) and n_d referring to number of degree, with $p_2 = n^{50}$ and $n_d = n^{10}$.

1 Introduction

This paper is all about converting an undirected graph $G = (V, E)$ to a signal processing problem, whether that be using a digital or analog computer, and use the reduction to resolve a $\#P$-complete problem - here the counting version of the hamiltonian path problem - efficiently/feasibly, at least if we interpret "feasible" as "polynomial time relative to $|V| = n$."

Graphs and functions have been studied together - for example, quantum graphs. In this line, this paper is one of these cases. However, it is surprisingly difficult to find an example where a graph is represented as a function and signal processing tools are used to analyze the function.

Many methods used in this paper are standard analog signal processing tools, and if there is anything novel in this paper, it would be about adapting the analog signal processing method to meet the numerical needs of the problem this paper intends to tackle.

Thus, the main reference for this paper is essentially analog signal processing books, or possibly even textbooks. I assume that the readers are familiar with signal processing terminology - terms like IIR, FIR, steady-state and transient response and so on. Or more precisely, the readers are assumed to be familiar with sinusoidal signal processing analysis.

2 Algorithmic approach

In this paper, all graphs are assumed to be undirected. Key primary insights are written in bold typeface, and secondary insights that are nevertheless of some importance will often be listed in itemized lists.

2.1 Two Goals

The first goal of this paper is to show that the counting variant of the hamiltonian path problem is effectively a signal processing problem. By reduction, this means that several important computation problems are "isomorphic" to signal processing problems.
The second goal of this paper is exploring whether feasible computation of the number of hamiltonian paths in an undirected graph is possible, via signal processing understanding of the hamiltonian path problem. For analog computation, the concept of "feasible" is obscure, but for digital computation, "feasible" will refer to computation in polynomial time.
For both cases, the answer is yes.

2.2 Two Core Concepts

These two core concepts are used to convert a hamiltonian path counting problem into a signal processing problem. The summary of core concepts is provided at the end of the next subsection, as part of Conclusion.

2.2.1 Core Concept 1 (CC1): reducing a graph to a function $f(t)$

As the title of this sub-subsection says, the first core concept is reducing an undirected graph to $f(t) : \mathbb{R} \to \mathbb{C}$.
Initially, one forms $x(t)$ before forming $f(t)$. **Each vertex is assigned a separate angular frequency.** Then each walk of a graph restricted to containing $|V| = n$ vertex visits is also assigned an angular frequency. By the definition of a walk, visiting one vertex more than once is allowed. Let us call such walks as n-walks for simplification.
Each n-walk can be assigned angular frequency by simply summing up angular frequency of each vertex by visit order. Thus, the angular frequency of a n-walk, expanded as the sum of vertex terms, may have one vertex added more than once.
The main reason for this function conversion is that in this process, all hamiltonian paths share the same angular frequency, because they have to contain all vertices in an undirected graph. Thus, one would like to have walks that are not hamiltonian paths to not share the angular frequency of hamiltonian paths. This is done by using particular angular frequency assigning methods so that each n-walk that shares the same set of vertices along with the same vertex-corresponding visit frequencies, has the same angular frequency. Of course this is not the only way, but this is the most straightforward

way of proceeding.
By visit frequency, if a n-walk visited some vertex v_a twice, then the visit frequency of v_a is 2.
Then I "shift" angular frequencies so that **hamiltonian path angular frequency is zero**, which forms $y(t)$. **This allows one to formulate a hamiltonian path counting problem as the signal processing problem of finding the period-adjusted average - or simply, the zero frequency amplitude - of the Fourier series** $y(t)$.
For convenience, I scale time/angular frequency, which gives one a Fourier series form of an input graph: $f(t)$.

2.2.2 CC1: more detailed conceptual explanations (CC1D1)

Definition 2.1 (visit frequency/occurrence)**.** Vertex v visit frequency/occurrence of a walk refers to the number of times v is visited in a walk. Visiting order does not matter.

Definition 2.2 (n-walk)**.** A n-walk is a walk with n vertex visits. That is for $G = (V, E)$, if one sums up visit occurrence of all vertices of a walk, one gets n.

Definition 2.3 (visit pair)**.** Visit pair for vertex v in a walk is the pair (v, k) that shows visit occurrence by natural number k. Visit pair of a walk is defined as the set of all such visit pairs, each corresponding to each vertex, for graph G. For example, for a graph with 4 vertices with $V = \{1, 2, 3, 4\}$, a walk may have visit pair of $\{(1, 2), (2, 0), (3, 1), (4, 1)\}$, where $4 = 2 + 0 + 1 + 1$.

A walk is assumed to be a valid walk allowed by undirected edge construction of G. The definition for a walk follows standard graph theory terminology.
The first motivation for constructing a grid representation $f(t)$ for a graph comes from the following list:

- A set of n-walks that share the same visit pair is assigned same angular frequency. That is, each walk is assigned angular frequency, but walks with a common visit pair have same angular frequency.
- Each n-walk has amplitude of 1. That is, each n-walk with angular frequency ω is said to contribute $e^{i\omega t}$.
- One wishes to ensure that a different n-walk visit pair is assigned a separate frequency, distinguishable from others.
- If the second item in this list is ensured, then it is possible to distinguish a n-walk visit pair that captures hamiltonian paths, where all n vertices of graph G exist, each with visit occurrence of 1.
- For final graph grid representation $f(t)$, we would like to make zero frequency hamiltonian path frequency. This is to extract the number of hamiltonian paths n_h using low-pass filtering techniques.

For this section, let U be the set of angular frequencies with corresponding n-walk in a graph.

Now consider how one assigns angular frequency to each walk.

Definition 2.4 (Vertex-number)**.** A vertex-number is assigned to each vertex. If not mentioned explicitly, vertex refers to a vertex-number it is assigned. In a particular implementation of the method used in this paper, each vertex is $v = n^i$ with $i \in \mathbb{Z}_+$. Thus, $V = \{n, n^2, n^3, .., n^n\}$.

From now on, $V = \{n, n^2, n^3, .., n^n\}$ is used.

Definition 2.5 (Assigning angular frequency to a n-walk)**.** Following from the visit pair p of a walk, with V in mind, recall the definition of p: $p = \{(v, k)|v \in V\}$, where k is visit occurrence. The angular frequency ω of each path is $\omega = \sum_v kv$. To state again, each walk contributes $e^{i\omega t}$.

Definition 2.6 (A_u)**. For this sub-subsection only**, A_u refers to the cardinality of the set of all the n-walks sharing angular frequency u. Separate definitions will be provided whenever A_u is used in different contexts.

Definition 2.7 ($x(t)$)**.** $x(t)$ be the sum of all n-walks.
That is, $x(t) = \sum_{u \in U} A_u e^{iut}$.

Definition 2.8 (a_h)**.** $a_h = \sum_v v$. That is, a_h refers to hamiltonian angular frequency for $x(t)$. For $y(t)$ and $f(t)$ this will not be the case.

Definition 2.9 ($y(t)$)**.** $y(t) = x(t)e^{-ia_h t}$. This shifts all angular frequencies of $x(t)$ by a_h, and hamiltonian angular frequency for $y(t)$ is now zero.

Definition 2.10 (maximum angular frequency)**. For this definition only**, I will relax the definition of A_u and U. U is assumed to be finite, and contains real numbers. Let A_u be some complex number that depends on u. Let some signal $k(t) = \sum_{u \in U} A_u e^{iut}$. Then maximum angular frequency of $k(t)$ refers to $|u|$ (notice the absolute sign, and this is not mistake) with greatest $|u|$ in the set U. The definition for maximum angular frequency applies for all parts of this paper.

Definition 2.11 (signal/part of angular frequency ω)**.** Consider $\mu(t) = \sum_\omega A_\omega e^{i\omega t} + \int_{\omega'} B_{\omega'} e^{i\omega' t}\, d\omega'$. Then signal/part of angular frequency ω of $\mu(t)$ is $(A_\omega + B_\omega\, d\omega)\, e^{i\omega t}$.

Definition 2.12 (amplitude at ω of some function $g(t)$)**.** It is $A_\omega + B_\omega\, d\omega$ mentioned in the definition for signal/part of angular frequency ω (right above).

Definition 2.13 ($f(t)$)**.** $f(t) = y(ct)$. c in this implementation is defined as $c = n^{(n^{11})}$. Thus $f(t)$ will have maximum angular frequency of $n^{(n^{11})+n+1}$, with minimum non-zero angular frequency (with non-zero amplitude) being $n^{(n^{11})}$. [In practice, maximum angular frequency of $f(t)$ will be lower, and this is over-estimate. Similarly, minimum non-zero angular frequency of $f(t)$ is under-estimate.]

This completes defining process of $f(t)$, which is to be used for low-pass filtering to obtain n_h. However, actual implementation of how to obtain $f(t)$ may be inferred but has not been presented yet. That is, we know that $f(t)$ is essentially the sum of different walks, but how will different walks be combined efficiently, so that we do not have to check individual walk to get $f(t)$ for some t?

2.2.3 CC1 continuation: efficient walk sum

For now, I will give analogy to a digital circuit that can easily be understood back in terms of algorithms. The idea is first given for $x(t)$, as calculation of $y(t)$ and $f(t)$ essentially follow automatically from calculation of $x(t)$.

- Recall that vertices were assigned vertex-numbers.
- Each vertex has two storage parts: incoming and outgoing storage.
- The idea now is to treat the entire graph as a digital synchronized/timed circuit.
- The digital circuit is assumed to be edge-triggered.
- Wires between vertices are connected to edge connectivity. If $(v_1, v_2) \in E$, then a direct wire connecting v_1 to v_2 exists. Note that we assume two wires exist for (v_1, v_2): one wire w_1 sends signal from v_1 to v_2, while the other wire w_2 sends signal from v_2 to v_1.
- w_1 is incoming wire to v_2, outgoing wire to v_1. w_2 is incoming wire to v_1, outgoing wire to v_2.
- When the digital circuit is triggered, each vertex first copies the signal in its incoming storage to outgoing storage. Then each vertex activates all the wires previously deactivated, so that signal may be transmitted.
- When the wires are activated, each vertex v receives signals from incoming wires, sum up all signals, multiply the sum by e^{ivt} and store the result to incoming storage.
- Right after the circuit is edge-triggered, each vertex also transmits signal stored in outgoing storage to outgoing wires.
- After new data are copied to incoming storage, all wires and the circuit are deactivated, waiting or the next trigger to occur.
- Before the first trigger, initial data stored in the incoming storage of each vertex v is e^{ivt}.
- After $n-1$th trigger time passes, one simply sums up incoming storages of all vertices and outputs $x(t)$.

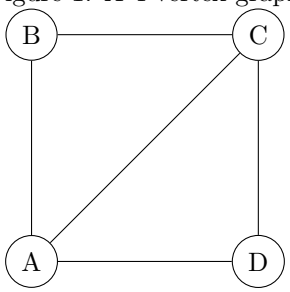
- As the last point, note that data in the storages will be vectors of polynomial coefficients that are obtained via Taylor-expanding around $t = 0$. Refer to the sub-subsection "The polynomial form of $f(t)$" for more details.

(At this point, one may be reminded of Boltzmann machines, though I will not go over this.)
Essentially, the idea above uses the following characteristic:

$$(e^{iv_{1,1}t} + e^{iv_{1,2}t})e^{iv_{2,3}t} = e^{i(v_{1,1}+v_{2,3})t} + e^{i(v_{1,2}+v_{2,3})t}$$

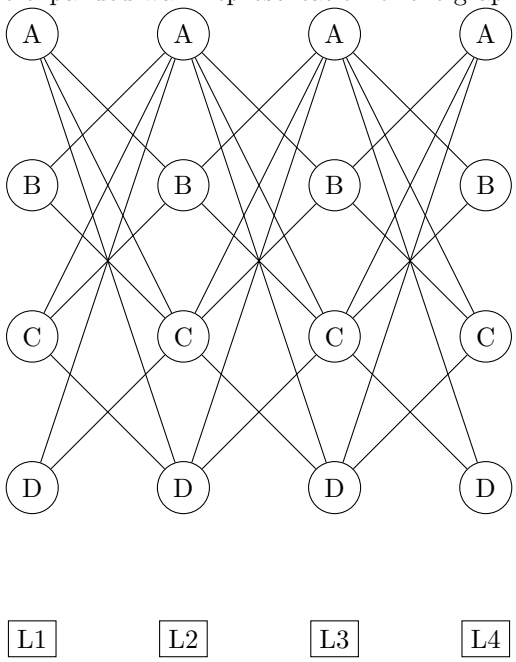
$v_{1,1}$ and $v_{1,2}$ can be thought of the first vertex visited for each corresponding n-walk. Both walks share the same second vertex visited, which is $v_{2,3}$. Notice that the first index (1 in $v_{1,2}$, for example) presents the visit order of a walk, while the second index distinguishes actual vertex. Thus, $v_{1,3}$ and $v_{3,3}$ represent the same vertex, but with a different visit order.
The full details will be given, but for now let us illustrate the principles using the example in Figure 1 and 2:

Figure 1: A 4-vertex graph



In Figure 2, because the original graph in Figure 1 is a 4-vertex graph, there are four layers or four depths, labelled with L1,L2,L3,L4. Each layer i contains all vertices in a graph. A vertex v in layer i is connected to a vertex w in layer $i + 1$ whenever $(v, w) \in E$. This grid procedure effectively simulates an actual walk.
At Layer 1 (L1), each vertex v transmits e^{ivt} to the edges, or wires, it is connected with. These edges connect to the vertices at the next layer L2. For other layers, each vertex v sums up all the function it received from the wires starting from the previous layer and multiplies the sum by e^{ivt}, and then transmits the product to the wires that connect v with the vertices of the next layer. The final layer L4 has an additional step, since there are no wires that connect to the next layer in Figure 2. Instead, all results obtained at each vertex at L4 is summed up, which results in $x(t)$.
To summarize, the vertices in L1 always act as oscillators, every edge that connects one layer to the next layer acts as a right-directional wire without any transmission delay and the vertices in each layer except L1 work first as a summer and then a multiplier coupled with an oscillator. After the final layer, a

Figure 2: The expanded walk representation of the graph in Figure 1.



summer adds up all results in the vertices in the final layer.
The constructed $x(t)$ does not have zero frequency as a hamiltonian path frequency and thus multiplication by $e^{-ia_h t}$, where a_h is the hamiltonian path angular frequency of $x(t)$, is needed to produce $y(t)$. Then for convenience angular frequencies may be scaled by multiplicative factor (just by changing time scale) to produce $f(t)$.
This completes introduction of the grid, and in fact, this is all there is for the grid.

2.2.4 CC1 implementation: full details

Definition 2.14 (Z^+, Z_+, Z^-, Z_-)**.** Z^+ or Z_+ refers to the set of positive integers. Similarly, Z^- refers to the set of negative integers.

Definition 2.15 ("less than", "more than", "greater than", "smaller than")**.** Unless otherwise noted, these are all comparisons in magnitude/size/absolute value.

Definition 2.16 (Base-n expansion)**.** Base-n expansion of some number k is basically expressing k in base-n: $k = \pm \sum_{p=-\infty}^{\infty} a_p n^p$ with $0 \leq a_p < n$.

Definition 2.17 (graph, n)**.** A graph G is denoted with $G = (V, E)$ as done in the standard literature. $n = |V|$ is assumed whenever n appears.

Definition 2.18 (walk, n-walk, hamiltonian path)**.** A walk is defined as in the standard graph theory vocabulary. A walk that has n vertices is called n-walk. Let us represent a walk with a list (tuple) of vertices in a traversing order from the start vertex to the end vertex. By the definition of a walk, one vertex can appear more than once in a list. A hamiltonian path, as defined in the standard graph theory vocabulary, is a walk with n distinct vertices, where $|V| = n$.

Definition 2.19 (vertex)**.** A vertex is assigned a number. Each distinct vertex has a distinct vertex-number. Let $V = \{n, n^2, n^3, ..., n^n\}$. From now on, one can assume a vertex as a number whenever appropriate.

Definition 2.20 (n_h, n_p)**.** n_h is the number of hamiltonian paths of G. n_p is the total number of n-walks of G.

Definition 2.21 (Walk-number)**.** The walk-number of a walk is defined as the sum of all elements (vertices) in the list of a walk. Note that "list" is used to refer to the fact a same vertex may be visited several times and it may need to be summed up several times.

Recall that the walk-number of a walk represents the angular frequency of a walk in $x(t)$. It is certainly possible that two walks may occupy the same frequency. If there are k walks that occupy the same frequency ω_a, then the amplitude at the frequency would be k in Fourier series language, or $k\delta(\omega - \omega_a)$ in Fourier transform language where $\delta(\omega)$ is a dirac delta function.
The maximum number of vertices inside a walk is restricted to n, for sake of convenience.

Lemma 2.1. *Given V as defined above, each walk-number has a unique visit pair attached. That is, no other visit pairs may generate that walk-number. (That each visit pair has a unique walk-number is trivial to prove.)*

Proof. The proof is simply the basis representation theorem, where basis are elements in V. One exception to this proof, though, arises when a list ξ representing a walk may be of $(k, k, ...k)$ with $|\xi| = n$ and $k = n^i$, or in words, there are n k's in ξ. In this case, $nk = n^{i+1}$, meaning the vertex-number ξ equals one of vertices in V. But this should not matter whenever walks one deals with have same number of vertices. □

Following from above:

Definition 2.22 (Contribution of each n-walk to $x(t)$)**.** From above, each walk has a walk-number k. Each n-walk is said to contribute e^{ikt} to $x(t)$.

Definition 2.23 (Amplitude)**.** For any arbitrary function $\alpha(t)$ expressible as $\alpha(t) = \sum_{\omega=-\infty}^{\infty} A_\omega e^{i\omega t/d}$ where d is constant and does not vary with ω, A_ω is said to be amplitude of $\alpha(t)$ at angular frequency ω.

2.2.5 CC1 continuation: grid: $x(t)$

Definition 2.24 (Grid, wires)**.** A grid consists of n depths, with each depth being equivalent to a column. Each depth contains n vertices as in V. Each wire connects a vertex v_α from ith depth to a vertex point of v_β in $i+1$th depth. A wire is connected between v_α to v_β if and only if $(v_\alpha, v_\beta) \in E$.

Definition 2.25 (Function transmission: first depth case)**.** In the first depth (first column), each vertex v_α transmits $e^{iv_\alpha t}$.

Definition 2.26 (Function transmission except for first and nth depth)**.** Defining for each v_α in arbitrary ith depth. All incoming wire transmissions $w_\zeta(t)$ from each wire ζ from $i-1$th depth to v_α in ith depth are summed, or equivalently $w_\lambda = \sum_\zeta w_\zeta$. And then multiply by $e^{iv_\alpha t}$ and transmit $u_{v_\alpha} = e^{iv_\alpha t} w_\lambda$ to each wire starting from v_α.

Definition 2.27 (Vertex point function transmission: nth depth case)**.** All incoming wire transmissions $w_\zeta(t)$ from each wire ζ from $n-1$th depth to v_α in nth depth are summed, or equivalently $w_\lambda = \sum_\zeta w_\zeta$. And then multiply by $e^{iv_\alpha t}$, resulting in $s_{v_\alpha} = e^{iv_\alpha t} w_\lambda$. $x_{ideal}(t) = \sum_{v \in V} s_v$ is the output of the grid, not considering polynomial coefficient errors and polynomial degree truncation involved.

For each depth i, $\sum_{v \in V} u_v$ shows the sum of all vertex-numbers representing i-walk.

2.2.6 CC1 continuation: post-grid: $y(t)$

Simply, this post-grid procedure is all about calculating $y(t) = x(t)e^{-ia_h t}$ where $a_h = \sum_{i=1}^{n} n^i$, the hamiltonian frequency of $x(t)$. Thus, $y(t)$ has 0 has hamiltonian frequency.

2.2.7 CC1 continuation: post-grid: $f(t)$

$f(t)$ is defined as $f(t) = y(ct)$. c was defined as $c = n^{(n^{11})}$.
Let the angular frequencies of $f(t)$ be labelled with ω. $\omega = 0$ refers to hamiltonian frequency.

2.2.8 The polynomial form of $f(t)$

The polynomial form of $y(t)$ is easily obtained by expanding e^{ivt} of each vertex v to the imposed polynomial degree $n_{d,1}$ (defined below) and do multiplication of the polynomial form of e^{ivt} (taylor expansion around $t = 0$) with the sum of incoming polynomials. The polynomial form obtained, truncated to degree $n_{d,1}$, is passed to outgoing wires. This gives us final accuracy up to polynomial degree $n_{d,1}$. There is coefficient error involved during the calculations, and this will be dealt separately.

2.2.9 Core Concept 2 (CC2): Taking advantage of known information: signal processing tools, multi-step

Definition 2.28 (polynomial term, polynomial form)**.** The polynomial term of degree d of some function $g(t)$ is the $A_d t^d/d!$ term that exists in the Taylor expansion of $g(t)$ around $t = 0$. The polynomial form of some function $g(t)$ of the assigned degree n_d refers to the truncated Taylor expansion of $g(t)$ around $t = 0$, where polynomial terms of degree greater than n_d are thrown out.

Definition 2.29 (term)**.** The term of some function will refer to the polynomial term with other terms also assumed to be polynomial terms, if not explicitly stated otherwise. When one says "$At^i e^{-t}/i!$ term," however, "term" may refer to transient response terms, and meaning will easily be inferred within contexts the word "term" is used. For example. the sentence like "$A_2 t^2 e^{-t}/2!$ term of function $g(t)$" may assume, depending on context, the decomposition/dissection of $g(t)$ into $g(t) = \sum_{i=0}^{d} A_i t^i e^{-t}/i!$. In other contexts, $g(t)$ may additionally include steady-state response, depending on contexts for the example sentence.

Definition 2.30 (A)**.** A does not really have a fixed meaning, and is used to represent an arbitrary number, amplitude and so on. Similarly with other letters without explicit definition like B, C, et cetera.

Definition 2.31 (function, t)**.** All functions are assumed to be of $\mathbb{R} \to \mathbb{C}$, with t being an independent variable.

Definition 2.32 (finite-degree truncation error)**.** Finite-degree truncation errors of the polynomial form of some $g(t)$ are the errors caused by Taylor-expanding $g(t)$ around $t = 0$ and truncating polynomial terms of degree greater than n_d (or $n_{d,1}$ in case of step 1) (defined below).

Definition 2.33 (n_d)**.** n_d refers to the degree of the polynomial form of the output polynomial, except step 1. That is, keeping the terms up to At^{n_d} terms, where A is context-relevant coefficient. $n_d = n^{12}$.

Definition 2.34 ($n_{d,1}$)**.** $n_{d,1}$ refers to the degree of the polynomial form of the output polynomial at step 1. The reason for separation between $n_{d,1}$ and n_d will be explained below. $n_{d,1}$ is also the imposed polynomial degree for $f(t)$.

Definition 2.35 (polynomial coefficient error)**.** Polynomial coefficient errors of the polynomial form of some $g(t)$ are the errors caused by finite-precision limit placed when storing the digits of polynomial coefficients of each polynomial term $A_k t^k$ of the polynomial form of some $g(t)$, with higher polynomial terms truncated. I will ignore this type of problem until the very end of this introduction section.

Definition 2.36 ($H(s)$, $H_r(s)$)**.** For $H(s)$, depending on context initially, it will refer to general linear filters, given in Laplace transform with variable s. However, for most of time $H(s) = 1/(s+1)$. $H_r(s) = -1/(s-1)$.

Definition 2.37 (step, step input)**.** A step sp contains filter $H(s)$, with the assigned initial condition 0 at $t = T_{sp,e}$ (defined below), along with filter input and output. There are two steps used in this paper, and the second step takes the polynomial input from the polynomial output of the first step. The first step takes the polynomial form of $f(t)$ as input. Step input does not refer to the heaviside step function input, but the input of some step sp.

Definition 2.38 ($T_{sp,e}$, r_1, r_2, r_{sp}, α)**.** $T_{1,e} \equiv r_1$, $T_{2,e} \equiv r_2$ and $T_{sp,e} \equiv r_{sp}$. $\alpha \equiv e^{r_1}$. The rule for setting α, r_{sp} is given in Equation 1 for all steps except $sp = 1, n_d + 2, n_d + 3$.

Definition 2.39 ($parity(i)$)**.** $parity(i)$ refers to even-odd parity of integer i. For odd integers, $parity(i) = -1$. For even integers, $parity(i) = 1$. Notice that the parity function is not 0/1-valued, but -1/1-valued.

Definition 2.40 (high-frequency steady-state input(s)/response(s))**.** High-frequency steady-state input(s) and response(s) refers to non-zero angular frequency parts of some given steady-state input/response. (Recall that steady-state responses are defined in terms of Fourier series.)

Definition 2.41 (plural form, part(s), input(s), output(s))**.** Recall that in the definition of high-frequency steady-state inputs, we used the plural form. We may have substituted the word "parts" equivalently. So why was the word "inputs" used? This is because by the linear time-invariant property of $H(s)$, it is better to think in terms of a single-angular-frequency signal separately, when analyzing responses to steady-state input. This is why we use the plural term, so I wish this does not cause any confusion here.

$$\alpha \frac{(r_{sp})^{sp-1}}{(sp-1)!} \sum_{i=0}^{n_d - sp + 1} \left[parity(i) \frac{t^i}{i!} \right] = 1 \tag{1}$$

For step 1, a rule concerning α must be set. Thus, once α is set, this determines all r_{sp}, via Equation 1, except r_{n_d+2} and r_{n_d+3}.

Definition 2.42 (step initial condition requirement)**.** Step sp initial condition (output-side) requirement is satisfied by the following: First, pass the polynomial form of step input to $H(s)$, resulting in $g_{sp}(t)$, and get the value $w_{sp} = g_{sp}(r_{sp})$. Expand e^{-t} to degree n_d, and substitute r_{sp}, and get value $e^{-r_{sp}} \approx z_{sp}$. Add the polynomial form of $w_{sp}e^{-t}/z_{sp}$ (to degree n_d) to $g_{sp}(t)$, which results in step output.

Definition 2.43 (steady-state response, transient response)**.** Steady-responses are the responses that can be written as Fourier series. Transient responses are the responses that cannot be written as Fourier series. These concepts are adapted from sinusoidal filtering literature, where transient response is the response that dies off to zero as $t \to \infty$. Here, transient response is defined more broadly - however, for purposes of this paper, these distinctions will not matter. A different definition is provided just for convenience of explanations

in this paper. In this paper, responses will refer to filter/step output, rather than input, unless explicitly noted, such as "input transient response." Input transient response refers to transient response being used as filter/step input.

Definition 2.44 (transient response/steady-state response restriction)**.** In this paper, I will restrict transient response to those that may have chance of being generated by allowed graphs (transforming to $f(t)$). That is, there should be some steady-state response (including $f(t)$ itself) input that generates transient response, and steady-state responses themselves must be allowed by the given input graph G.

Definition 2.45 (steady-state input)**.** This refers to steady-state response being used as filter/step input. As said in the definition of steady-state response, if not noted otherwise with label "input," steady-state response refers to filter/step output.

Definition 2.46 (transient response induced/caused/created (three words equivalently used) by steady-state input)**.** This will refer to step output caused by steady-state input, as the name implies. If we talk about transient response at step sp induced by steady-state input at sp' where $sp' < sp$, this refers to the following. Step sp' steady-state input causes transient response Ae^{-t}. At step $sp' + 1$, suppose step input is Ae^{-t} only. Then use the resulting step output of step $sp' + 1$ (which is transient response) for step input of step $sp' + 2$, and continue on until step sp. It is this step sp output (assuming Ae^{-t} step input at step $sp' + 1$) that we call transient response at step sp induced by steady-state input at step sp'. That is, transient response induced by steady-state input neglects the effects of steady-state responses all together from step sp', and deals only with what happens by transient response of steady-state input at step sp', which pipe into step sp.

Definition 2.47 (Integrative nature of IIR filters)**.** Consider $H(s) = 1/(s+1)$ $H(s)$ can be dissected into: $H(s) = 1/s - 1/s^2 + 1/s^3 - 1/s^4 + ...$ Notice how $H(s)$ is essentially a sum of cascades of integrators. This means that one does not have to worry about polynomial coefficient errors, as long as there is no input polynomial coefficient error, when one does not consider the initial condition change of the step (defined above).

Now let us think about filtering $f(t)$ so that one may obtain zero-frequency amplitude n_h. Naively just using $H(s)$ would result in huge error. This is because, angular frequency ω is very large that finite-degree truncation error will be large. Relying on samples (and digital filters) is not feasible, as they basically have the same problem - one can see this clearly by considering the polynomial interpolation of these samples.
Thus, a clever mechanism is required, and this paper intends to provide that one. The integrative nature of IIR filters would prove to be very useful.

2.2.10 CC2: clever mechanism

Let us evaluate how α must be set, and why Equation 1 was set. Because of the flow of the discussion, in case any confusion arises regarding $n_{d,1}$, please refer to sub-subsection 2.2.17.
Equation 1 is the truncated version of the following equation:

$$\alpha \frac{(r_{sp})^{sp-1} e^{-r_{sp}}}{(sp-1)!} = 1 \tag{2}$$

This comes from the idea that we would like to eliminate Ae^{-t} terms when the first step input is $k_0 \in \mathbb{R}$. Consider the first step output with the input k_0:

$$o_1(t) = k_0 - k_0 \alpha e^{-t} \tag{3}$$

where α is set by imposing the zero initial condition at $t = r_1$. Pass this output to the second step:

$$k_0 + \gamma e^{-t} - k_0 \alpha t e^{-t}$$

One would like to set γ as zero. Separate $o_1(t)$ into k_0 and $-k_0 \alpha e^{-t}$. What Equation 2 says is that the Ae^{-t} term of the second step output to $-k_0 \alpha e^{-t}$ must be cancelled out by Ae^{-t} term of the second step output to k_0. That is, at $t = r_2$, one should make $-k_0 \alpha e^{-t} = -k_0$ at $t = r_2$ so that it is cancelled out by k_0, which means the zero initial condition at $t = r_2$ is satisfied without creating additional transient responses. And so forth with other steps.
So far, I have considered the case with the first step input only being $k_0 = n_h$. But we also have high-frequency steady-state inputs/outputs, which need to be considered.
The fundamental choice being made here is this: **we will allow the first step transient response to high-frequency steady-state response to be extremely large coming mostly from "numerical errors." That is, we will set r_1 and α to be relatively large.** This by Equation 1 implies a very small r_{sp} for steps other than the first step, which also means there is very small numerical error. By numerical error, it refers to the error in calculating the value at $t = r_{sp}$ of the step input passed to $H(s)$ due to the finite (polynomial) degree restriction of the input.
Now recall the n_d degree limit. Consider again input k_0. This means that at step $n_d + 1$, the output is:

$$k_0 - \frac{k_0 \alpha t^{n_d} e^{-t}}{n_d!}$$

but e^{-t} here serves effectively as 1. Let us pass this output as an input to filter $H(s)$ and we see that $k_0 \alpha t^{n_d} e^{-t}/n_d!$ disappears completely, and only k_0 remains the relevant input for the output. This is an important point, for we can define a cycle consisting of $n_d + 1$ steps.
After finishing the cycle, we have to re-define r_{n_d+2}, as Equation 1 no longer applies. Before doing so, let us consider what happens at the input of step $n_d + 2$. Let us define r_μ, $r_{\mu+1}$:

Definition 2.48 $(r_\mu, r_{\mu+1}, \mu)$**.** $r_\mu = r_{n_d+2} \equiv r_{(n_d+2)}$, $r_{\mu+1} = r_{n_d+3}$. $\mu = n_d+2$.

Assume that step μ input can be approximated by the following:

$$\nu(t) = k_0 + \sum_{i=0}^{n_d} \frac{k_1 t^i e^{-t}}{i!} = k_0 + \Pi_a \tag{4}$$

We will return back to the question of whether the approximation is useful for our analysis.
Let us again divide the input approximation into two inputs: k_0 and Pi_a. First consider k_0. The output by passing k_0 to step μ is:

$$k_0 - k_0 \beta e^{-t}$$

which means $\beta = e^{r_\mu}$, or more correctly approximately with the right choice of r_μ and n_d. Pass this to step $\mu + 1$ and specify the desired output:

$$k_0 - k_0 \beta t e^{-t}$$

Notice again that intention is eliminating e^{-t} term. This is done by:

$$\frac{e^{r_{\mu+1}}}{r_{\mu+1}} = \beta = e^{r_\mu} \tag{5}$$

(with the right choice of n_d and r_μ and $r_{\mu+1}$.)
Let us specify the step $\mu + 1$ output, given the approximation input $\nu(t)$:

$$k_0 - k_0 \beta t e^{-t} + \Pi_b + \Pi_c + \Pi_d + \Pi_e \tag{6}$$

where $\Pi_b, \Pi_c, \Pi_d, \Pi_e$ are defined by:

$$\Pi_b = -k_1 \left[tr\left(e^{r_\mu}\right) - 1 \right] t e^{-t} \tag{7}$$

$$\Pi_c = k_1 \left[tr\left(e^{t}\right) - 1 - t \right] e^{-t} \tag{8}$$

$$\Pi_d = k_1 \left[tr\left(e^{r_\mu}\right) - 1 \right] r_{\mu+1} e^{-t} \tag{9}$$

$$\Pi_e = -k_1 \left[tr\left(e^{r_{\mu+1}}\right) - 1 - r_{\mu+1} \right] e^{-t} \tag{10}$$

and $tr(e^{r_{sp}})$ refers to truncating e^t to degree n_d and substituting $t = r_{sp}$, and similarly with $tr(e^t)$, which truncates e^t to degree n_d but without substitution.
Let us provide a system of equations of the constant term (c_0) and At term ($c_1 t$, $A = c_1$) of step $\mu + 1$ polynomial output:

$$\begin{array}{rcl} k_0 + k_1 \left(e^{r_{\mu+1}} - \frac{e^{r_{\mu+1}}}{r_{\mu+1}} + 1 \right) & = & c_0 \\ -k_0 \frac{e^{r_{\mu+1}}}{r_{\mu+1}} - k_1 \frac{e^{r_{\mu+1}}}{r_{\mu+1}} & = & c_1 \end{array} \tag{11}$$

Solving the system of equations results in:

$$k_0 = \frac{c_0}{\beta - e^{r_2}} + c_1 \frac{e^{r_{\mu+1}} r_{\mu+1} - e^{r_{\mu+1}} + r_{\mu+1}}{e^{r_{\mu+1}}} \frac{r_{\mu+1}}{e^{r_{\mu+1}} - r_{\mu+1} e^{r_{\mu+1}}} \tag{12}$$

$$n_h \equiv k_0 \approx \frac{c_0}{\beta} - \frac{c_1}{\beta} \tag{13}$$

To derive Equation 13 from Equation 12, this of course requires the condition that $\beta \gg 1$. Double-approximative equation 13 captures the dominant terms in denominators in Equation 12.

2.2.11 CC2: role of Equation 12 and 4

Definition 2.49 (Filter-H-initial value)**.** Recall the definition for a step. A step first takes filter input, pass it to $H(s)$ and impose the initial condition. Let us eliminate the initial condition and think as if we passed the step input to filter $H(s)$. Then filter-H-initial value of step sp refers to the step output without imposed initial condition at $t = r_{sp}$.

Before going on, let me emphasize that **Equation 12 will not be used for solving k_0.** After all, there is danger that inaccurate $k_0 = n_h$ may be obtained. So what is Equation 12 actually giving us, and what exactly is the approximate input $\nu(t)$ in Equation 4?
Let us come back to step 2. Suppose step 1 input is $k_x e^{-t}$. Then passing this to step 2 results in:

$$-k_x r_2 e^{-t} + k_x t e^{-t}$$

assuming numerical error in computing filter-H-initial value due to the polynomial approximation is small enough. Notice that $r_2 < 1$. For step 3, the output becomes:

$$k_x \left(-\frac{(r_3)^2}{2} + r_2 r_3 \right) e^{-t} - k_x r_2 t e^{-t} + \frac{k_x t^2 e^{-t}}{2}$$

Consider how $r_2 r_3$ and $(r_3)^2/2$ compares. Essentially, $(r_3)^2 \approx r_2$, as both $r_2, r_3 \ll 1$. Thus, $r_2 r_3$ is much less than $(r_3)^2/2$. Continue these processes until step $n_d + 1$. **It is now seen that for step μ, $\nu(t)$ is created assuming that for step $2 \leq sp < \mu$, Ae^{-t} term of step sp output was affected only by the $Bt^{sp-1}/(sp-1)!$ term of step sp output.** By "approximative" relation: $r_2 \approx (r_d)^{1/(d-1)}$ for step $d \leq n_d + 1$, we can see that the actual step μ input's magnitude is $(1 + \mu_d)|\nu(t)|$, where $\mu_d < 2^{n_d} r_{n_d+1}$.
Thus now we have obtained the assumption: $1/r_{n_d+1} \gg 2^{n_d}$, which is to be used with μ_d to figure the effects of the parts truncated from actual step μ input by using $\nu(t)$.
System of equations is now given as:

$$\begin{array}{rcl} k_0 + (1+\mu_d)k_1 \left(e^{r_{\mu+1}} - \frac{e^{r_{\mu+1}}}{r_{\mu+1}} + 1 \right) & = & c_0 \\ -k_0 \frac{e^{r_{\mu+1}}}{r_{\mu+1}} - (1+\mu_{d'})k_1 \frac{e^{r_{\mu+1}}}{r_{\mu+1}} & = & c_1 \end{array} \tag{14}$$

Note the added μ_d and $\mu_{d'}$. Solve the equation and one sees that the dominant term equation, which is Equation 13 remains valid.
Now coming to the question of why Equation 13 is useful. **Recall that we have not so far considered steady-state response at step $\mu + 1$ and transient responses caused by high-frequency steady-state inputs of step 2 and beyond.** What Equation 13 allows us is to check whether they do matter.
It is very easy to see that they do not. Again, recall Equation 13

$$n_h = k_0 \approx \frac{c_0}{\beta} - \frac{c_1}{\beta}$$

The polynomial term of degree n_d of the steady-state response of high-frequency ω at step $\mu+1$ has the following (with denominator only capturing the dominant term):

$$A\frac{\omega^{n_d}t^{n_d}}{\omega^{n_d+2}}$$

Thus, it is clear that we have the right degree n_d at step $\mu+1$ (this indeed is somewhat tautology via definition, but worth emphasizing in numerical concerns). For our relevant constant term and At term, this is more than sufficient. As the contribution to c_0 is much less than 1, our calculation of n_h is not affected. Similarly with c_1 also. I will revisit this again, along with transient response analysis associated with these steady-state responses.
Now contributions of transient responses not considered. The dominant contribution comes from step-2 transient responses induced by step 2 steady-state input (or step 1 steady-state response/output, equivalently). But $e^{r_\mu}2^{n_d}n^n/(\omega)^2$ can easily be made to be insignificant by choosing the right r_μ and c in $f(t) = y(ct)$ so that these transient responses cause less than 1 contribution in magnitude to c_0 or c_1.
This completes our analysis (it may seem incomplete here, but the above subsubsections will show that it is indeed complete) except the issues involving step μ and step $\mu+1$ transient responses induced by step μ and $\mu+1$ steady-state inputs, and requirements will be listed below.

2.2.12 Numerical concerns, requirements

- $n_{d,1} \gg n_d$, $n_{d,1} \gg r_1$ and $n_d \ll r_1$
  - so that $2^{n_d}n^n r_{n_d+1} \ll 1$ requirement may be satisfied: this requirement deals with $\nu(t)$ and Equation 13 so that analysis based on them can show that post-step-1 transient responses induced by post-step-1 steady-state inputs and steady-state inputs themselves do not matter for our calculation of $n_h = k_0$.
  - so that we may be able to shrink r_{sp} for $2 \le sp \le n_d+1$ to satisfy $\omega r_{sp} \ll 1/n^n$ in magnitude.
- $e^{r_\mu}2^{n_d}n^n/(\omega)^2 \ll 1/n^{2n}$ so that post-step-1 steady-state inputs have do not create transient responses affecting our value of c_0 and c_1 significantly.
- $\beta \equiv e^{r_\mu} \gg 1$ so that Equation 13 becomes valid, and so that above items are satisfied without a problem.
- Finally, $r_\mu \ll n_d$ so that filter-H-initial value error caused at step μ is insignificant.

2.2.13 Again, numerical concerns and other numerical concerns

So far, we have not discussed how insignificant filter-H-initial-value errors are, given our r_{sp}, $n_{d,1}$ and n_d. This is because they truly are very insignificant.

The below addresses that issue, plus provides a detailed summary of what was presented above.

- Step 1 filter-H-initial-value errors do not matter. This is because k_1 already incorporates these errors by assuming the actual A in the Ae^{-t} term at step 1.
- Step 2 to step $n_d + 1$ to step $\mu + 1$
  - Step 1 input k_0 always induces zero error: the rules for setting r_{sp} ensure this.
  - We will not consider the fact that a cycle of n_d+1 steps is formed due to polynomial truncation itself as being an error. What we consider as filter-H-initial-value errors are mainly related to e^{-t} term truncations. The exception is when we used e^{r_μ} instead of the truncated one.
  - Let $\omega r_{sp} \ll 1/n^n$ for $2 \leq sp < \mu$. Let $n_d = n^{10}$, and let $r_2 = n^2$. Then, filter-H-initial value errors magnitude upper bound at step $\mu + 1$ can be considered as $2^{(n_d+3)}(n_d + 3)e^{r_2}n^n/n^{(n^{11})} \approx 1/n^{(n^{11})}$, for sum of post-step-1 transient responses induced by post-step-1-until-step-$n_d + 1$ high-frequency steady-state inputs.
  - Steady-state responses have zero error, via the integrative nature of $H(s)$. (IIR filter)
  - The filter-H-initial value errors magnitude upper bound at step μ and $\mu + 1$ induced by step μ high-frequency steady-state inputs are meaningless. Instead, we rather would have to consider in terms of actual calculated steady-state value at $t = r_\mu$ at step μ.
  - What was discussed in the above item applies for the filter-H-initial value errors magnitude upper bound at step $\mu + 1$ induced by step $\mu + 1$ high-frequency steady-state inputs.
  - To summarize the condition above and below, $e^{r_\mu}2^{n_d}n^n/\omega^2 \ll 1$ for high-frequency ω.

2.2.14 High-frequency steady-state input/output at step μ and $\mu+1$ and associated transient responses

When $\omega r_{sp} \ll 1$ is not satisfied, as is the case with step μ, it is possible that filter-h-initial-value is extremely large for high-frequency $|\omega| \gg 1$. Fortunately, this is not the case. Recall the following polynomial term of degree k (denominator only capturing the dominant term) for the step μ high-frequency steady-state output:

$$A\frac{\omega^k t^k}{\omega^{n_d+2}} = A\frac{t^k}{k!\omega^{n_d+2-k}}$$

where $|A| \leq n^n$. Simplifying analysis, one may form the upper magnitude bound of the calculated polynomial high-frequency steady-state output as (with

appropriate $r_\mu = n^{\psi_r}$, with $(r_\mu)^{r_\mu}/(r_\mu)! \approx n^{r_\mu}$:

$$\frac{n_d n^{r_\mu} n^n}{\omega^2} \tag{15}$$

We would like to make the above satisfy:

$$\frac{n_d n^{r_\mu} e^{r_\mu} n^n}{\omega^2} \ll 1 \tag{16}$$

But this is satisfied by the constraint/requirement related to step-2 steady-state-input-induced transient response:

$$\frac{e^{r_\mu} 2^{n_d} n^n}{\omega^2} \ll 1 \tag{17}$$

As long as the above equation is satisfied, step $\mu + 1$ steady-state input also is insignificant for our calculation of n_h. Equation 16 can also be used for step $\mu + 1$, with transient response at step μ being used as input for step $\mu + 1$. This is because $r_{\mu+1} \ll 1$.

2.2.15 CC2: implementation values

Let us refer back to the two lists (List 2.2.12 and 2.2.13) above. Let $n_d = n^{10}$, $r_\mu = n^2$. Let us satisfy Equation 17 first (also the last item in List 2.2.13). Let us choose $c = n^{(n^{11})}$ in $f(t) = y(ct)$. This gives us angular frequency range of $n^{(n^{11})}$ to $n^{(n^{11})+n+1}$ in positive or negative angular frequency directions, plus of course zero frequency, which hosts our hamiltonian path angular frequency. With this chosen, let us choose r_1, which by reference to n_d also allow us to restrict $n_{d,1}$. Let $r_1 = n^{30}$, which is set with consideration that $1/(n^{(n^{29})})^{1/n^{10}} = 1/n^{(n^{19})}$, which gives us the range of r_{sp} for $2 \leq sp < \mu$. This choice of r_1 satisfies the requirement that $2^{n_d} n^n r_{n_d+1} \ll 1$ (the fifth item, or the third sub-item connected to the second item, of List 2.2.13 and the second item, or the first sub-item connected to the first item, of List 2.2.12). Together, these satisfy the third and fourth item of List 2.2.12. The fifth item and the sixth item of List 2.2.12 was satisfied via the choice of n_d and r_μ.
This leaves us with the choice of $n_{d,1}$. Let us choose $n_{d,1} = n^{40}$.
Let us summarize:

$$\begin{aligned} n_{d,1} &= n^{40} \\ n_d &= n^{10} \\ r_1 &= n^{30} \\ r_\mu &= n^2 \\ f(t) &= y(n^{(n^{11})}t) \end{aligned} \tag{18}$$

We are really done in analysis.

2.2.16 CC2: extraction of n_h

Note that while Equation 12 was useful in showing significant contributions that must be used to calculate $k_0 = n_h$, it nevertheless does not by itself accurately

compute n_h.
But one can easily see that the way to compute n_h then is as follows: have e^{-t} as (pseudo-)step two input and run all the steps using all the defined r_{sp} and imposed $n_{d,1}$, n_d.

Definition 2.50 (pseudo-step)**.** Pseudo-step is defined as to distinguish with actual step operations. Pseudo-step does not have pseudo-step 1: pseudo-steps start with pseudo-step 2. The input is always e^{-t} at pseudo-step 2. Pseudo-step $\mu + 1$ output is used to determine $\phi_{0,1}$ and $\phi_{1,1}$ below. Thus, pseudo-steps are only used to determine coefficients in the final system of equations used to solve $n_h = k_0$.

This gives us a complete picture on the relationship of $k_0\phi_{0,0} + k_1\phi_{0,1} = c_0$ and $k_1\phi_{1,0} + k_1\phi_{1,1} = c_1$. $\phi_{0,0}$ and $\phi_{1,0}$ was already known (and equivalent as in Equation 11, so the problems that get resolved here is on $\phi_{0,1}$ and $\phi_{1,1}$. Algorithm 4 provides the implementation. In the implementation k_1 is replaced with z_1 to give the system of equations:

$$\begin{array}{rcl} k_0\phi_{0,0} + z_1\phi'_{0,1} & = & c_0 \\ k_0\phi_{1,0} + z_1\phi'_{1,1} & = & c_1 \end{array} \tag{19}$$

where z_1 refers to step 2 transient response input z_1e^{-t} created by high-frequency steady-state inputs at step 1.

2.2.17 Revisiting $n_{d,1}$

From the discussion, it may not be clear why $n_{d,1}$ must be different from n_d. This all comes from the fact that when e^{-t} is expanded and substituted with r_1, adding higher-polynomial-degree polynomial terms of e^{-t} makes the calculated value diverge from actual e^{-r_1} value before reaching a certain degree. And it is the calculated e^{-t} that is used to determine A in Ae^{-t} to be added to the filter output, given step input. When $n_{d,1}$ is around r_1 or less than r_1, then calculated e^{-r_1} value is far off from actual e^{-r_1} value. And because of this explosion, we see that big r_1, given insufficient n_d actually shrinks α in Equation 3. This means r_2 and other r_{sp}s are affected, which now need to be much greater, causing large deviation in steady-state value (filter-H-initial value error).
This is why we set $n_{d,1}$ to be much greater than n_d.

2.2.18 For both Core Concepts: finite coefficient precision issue and implementation value of p_1, p_2

So far, I have discussed the effects caused by the polynomial form of functions truncating polynomial terms of degree greater than n_d. But polynomial coefficients used for functions also need to be stored in finite and reasonable amount of digits.
The idea here is incredibly simple. Recall our system of equations for c_0 and c_1 above. The equations say that as long as polynomial coefficient precision is relatively big enough, additional factors/deviations in c_0 and c_1 are too insignificant to values of k_0 and k_1.

Definition 2.51 (polynomial coefficient precision: p_1, p_2)**.** Polynomial coefficient precision digits. These are set to be more than sufficient. $p_1 = n^{60}$ binary digits (precision imposed on $f(t)$ and grid intermediate calculations), $p_2 = n^{50}$ binary digits (Extraction/filter interemediate and final results).

2.3 Algorithmic approach conclusion and time complexity analysis

2.3.1 Core Concept Summary

- A given undirected graph $G = (V, E)$ was converted to $f(t)$. Then, we formulated the signal processing problem as zero-frequency amplitude extraction of the Fourier series $f(t)$. This zero-frequency amplitude gives one n_h, the number of hamiltonian paths.
- We directly work on the polynomial approximations of functions, instead of using samples. This makes us choose analog IIR filters, as it provides precise polynomial coefficients for filter outputs, regardless of truncation degree chosen, as long as input polynomial coefficients are accurate. By the polynomial approximations, it refers to Taylor expansion around $t = 0$, truncated to degree n_d. This truncation degree is fixed for every filter output and input.
- The main idea used in this paper is about eliminating Ae^{-t} term when constant term was the input. That is, suppose $f(t) = n_h$. Then for every filter output, we would like to eliminate the effects of transient response at $t = 0$, which means eliminating Ae^{-t}. But notice that when one passes n_h to filter $H(s)$, transient response is inevitable. Thus we instead choose to eliminate the effects after one filtering operation (step) is completed, meaning Ae^{-t} effects are eliminated at the second step. (Each step uses filter $H(s)$ with atypical initial condition.) That is, we filter several times - which means implementing high-order filters as a cascade of first-order filters, but with atypical initial condition. (Typical initial condition is filter output being zero at $t = 0$. We instead choose to make filter output zero at $t = r_{sp}$, where sp refers to step, or the ordinal of the filter in the cascade of filters used.
- We of course have to consider high-frequency steady-state inputs of $f(t)$. But eliminating high-frequency steady-state input-induced transient response effects at $t = 0$ is difficult (as we do not know exact ω) and is not done. Instead, via clever use of r_{sp} and n_d, we form a system of equations so that we may obtain n_h.
- What happens in Core Concept 2 is incredibly simple, even though details are not. Polynomial truncation at degree n_d means that output to input transient response $t^{n_d}e^{-t}/n_d!$ disappears, meaning that $n_d + 1$ steps form a cycle. Equation 2 shows that if r_1 is very big, then r_2 is very small. Since we would like the polynomial truncation effects of steady-state responses

to be small for many steps as possible, this is the way we choose. This allows one to control transient response magnitude, except for the first step.

- Step μ and $\mu + 1$ work essentially in the same spirit as step 1 and 2, except that now we do not need to set μ to be very large. This is because steady-state response decayed significantly to allow us to take benefits of polynomial truncation. (We often think polynomial truncation only causes numerical errors, and these errors are in fact what this paper tried to overcome. But at step μ and $\mu + 1$, they rather come as advantage in that we really do not wish high-frequency steady-state information and want them to be completely ignored as zero.)

- Why $n_{d,1}$ is set is explained again in the separate sub-subsection and will not be mentioned again. The sub-subsection provides an insight on how n_d is set, along with other sub-subsections, and will be mentioned again.

Another Core Concept 2 summary (a very short one without explanations):

- The point basically is to set a system of equations that involves only $n_h = k_0$ and k_1 in the first-step-high-frequency steady-state input-induced transient response $k_1 e^{-t}$. This requires setting r_{sp}, n_d and $n_{d,1}$ so that other types of transient responses not caused by the first step steady-state input can be ignored.

2.3.2 Time complexity analysis

Even without going into deep analysis, it is clear by now that the above method provides a polynomial time algorithm for computing n_h. But for sake of completeness, let us perform time complexity analysis.
First, consider what time complexity would be like for filtering process. We used polynomial forms for filter inputs and outputs. And $t^k/k!$ in Laplace domain is $1/s^{k+1}$, which is a cascade of integrators.
Thus, relative to the length of analysis we have done so far, actual filtering process is ridiculously simple.
Let the temporary storage be the polynomial $u(t)$, initialized with zero. Let $p(t)$ be the polynomial form of inverse laplace transform of $1/(s+1)$ to degree n_d, and $p_1(t)$ be the polynomial form of inverse laplace transform of $1/(s+1)$ to degree $n_{d,1}$. For now let us consider normalized polynomial coefficients ($B = Ak!$ in At^k). Some step polynomial term $B_k t^k/k!$ translates to simply moving the each normalized polynomial coefficient of the polynomial form $p(t)$ by $k+1$ to the right and multiplying these normalized coefficients by B_k. Then add the resulting coefficients to $u(t)$. Continue the process from $k = 0$ to $k = n_d$ (or $k = n_{d,1}$ in case of step 1) of step input. Then, get $u(r_{sp})$, and add the polynomial form of $-u(r_{sp})p(t)/p(r_{sp})$ to $u(t)$. ($p_1(r_{sp})$ and $p_1(t)$ instead of $p(r_{sp})$ and $p(t)$ in case of step 1)
Thus, for complexity purpose in O-time, time complexity is $O(n_d(n_{d,1})^3(p_2)^2)$,

with $(p_2)^2$ referring to multiplication of two p_2-digit numbers, $(n_{d,1})^2$ refers to the fact that there are at maximum $n_{d,1}$ polynomial terms that do integration, or polynomial coefficient shift, that have multiplicative B_k of the step input to be multiplied, and $n_{d,1}$ polynomial terms at maximum that the B_k multiplicative term is multiplied to. $O(n_d + 3)$ refers to number of steps being used. Another $n_{d,1}$ refers to the number of multiplications for t^k terms (used when determining the value at $t = r_{sp}$), with $k = n_{d,1}$ at maximum (exponentiation). To calculate, $O(n^{230})$. Note that for simplicity, we actually derived time complexity that is much more than actual time complexity. In fact, the parameters used for this paper could have been less.
Now consider the grid procedure. Time complexity is $O(n^2 \cdot (n_{d,1})^2 (p_1)^2)$, which refers to the dominant e^{ivt} multiplication part of the inner loop of the grid. See the Algorithm 1 for reference. n comes from the outer loop, the other n comes from the inner loop, excluding the inner loop of the inner loop (used for coefficient additions). $n_{d,1}$ refers to the times/number of terms each polynomial coefficient of e^{ivt} needs to be multiplied to, the other $n_{d,1}$ refers to the number of such polynomial coefficients of e^{ivt}. And $(p_1)^2$ of course refers to multiplication of two p_1-digit numbers. This results in $O(n^{202}) \approx O(n^{210})$.
So far, total time complexity was $O(n^{230})$, but note again that we have so far selected sufficient parameters, not efficient parameters.
Now time complexity for determining r_{sp}. Determination is done by Equation 1, which gives us a problem of solving the polynomial equation equalling to zero - that is find the root of the polynomial equation. There are at maximum $O(n_d)$ polynomial terms, plus coefficients are $O(n^{50})$-digits. Since there are many polynomial equation solvers that are of polynomial time to n_d and $p_1 = n^{50}$ (see, for example, [2]), I will simply denote time complexity as $PC(n_d, p_1)$.
Now comes the extraction (of n_h) part. But this part provide relatively insignificant time complexity. It was noted in sub-subsection 2.2.16 that the equation and essentially the pseudocode we use is this:

$$\begin{aligned} k_0\phi_{0,0} + k_1\phi_{0,1} &= c_0 \\ k_0\phi_{1,0} + k_1\phi_{1,1} &= c_1 \end{aligned} \tag{20}$$

(with of course, modification as in Equation 19, but this really does not matter.)
And solving the system of equations is obviously pure multiplication and subtraction and is largely insignificant. Rather, what is significant is determining $\phi_{0,1}$ (or more precisely, $\phi'_{0,1}$ in Equation 19) and $\phi_{1,1}$ (or more precisely, $\phi'_{1,1}$). But this is simply running step 2 input e^{-t} until step $\mu + 1$, which gives us $\phi_{0,1}$ and $\phi_{1,1}$. Thus, total time complexity derived is $O(n^{230}) + O(n_d PC(n_d, p_2))$, which is polynomial-time complexity.

2.3.3 Drawing the result from limited additional information

It may seem at first that because there are exponentially many angular frequencies considered with equal angular frequency, there is no way one can efficiently draw out n_h.
However, we do not need accurate information of amplitudes of $f(t)$ at angular

frequencies other than zero. Furthermore, we know where maximum angular frequency with non-zero amplitude possibly lies, and where minimum non-zero angular frequency with non-zero amplitude lies. We also know that maximum sum of amplitudes and each amplitude can only be of magnitude n^n at maximum.
Just like compressed sensing uses limited additional information - for example assumption of sparsity - to draw out results, limited information does allow us to get the desired result n_h.

2.4 Pseudocode for grid

The pseudocode is provided by Algorithm 1.

Algorithm 1: Grid procedure for calculating $f(t)$

Input: Graph $G = (V, E)$, coefficient precision $p_1 = n^{60}$
Output: The polynomial form of $f(t)$ to degree n_d within coefficient precision p_2
$n \leftarrow |V|$;
Let *depth* be d, $V[l]$ be lth vertex, *wire* be w;
$w[l][d]$ is a vector of polynomial coefficients for outgoing wires from depth d vertex $V[l]$ to $d+1$;
d starts at 1, ends at n. At $d = n+1$, the algorithm stops;
e^{ivt} stands for the vector of polynomial coefficients up to polynomial degree n_d;
e^{ivt}'s polynomial coefficients are determined by Taylor expansion around $t = 0$;
$V[l] == n^l$, where l goes from 1 to n, with V matches to vertices of G;
$z * e^{ivt}$ refers to multiplication of the polynomial z by the polynomial form of e^{ivt};
$z * e^{ivt}$ also truncates the multiplication result to polynomial degree n_d;
When calculating $z * e^{ivt}$, polynomial coefficient is calculated to p_1 digits;
$a_h \leftarrow \sum_{l=1}^{n} V[l]$;
initialize $w[l][1] = e^{i(V[l])t}$;
$d \leftarrow 2$;
repeat
  repeat
    $j \leftarrow 1$;
    repeat
      if $(V[j], V[l]) \in E$ **then**
        $w[l][d] \leftarrow w[l][d] + w[j][d-1]$;
      $j \leftarrow j + 1$;
    until $j > n$;
    $w[l][d] \leftarrow w[l][d] * e^{iV[l]t}$;
    $l \leftarrow l + 1$;
  until $l > n$;
  $y(t) \leftarrow \left(\sum_{l=1}^{n} w[l][n]\right) * e^{-ia_h t}$;
  $f(t) = y(n^{(n^{11})}t)$;
  $d \leftarrow d + 1$;
until $d > n$;

2.5 Pseudocode for multi-staged/multi-step filtering

The algorithm is provided in Algorithm 2. The algorithm for r_{sp} is the polynomial root solving, presented in Equation 1. Normalized polynomial coefficient refers to A in $At^k/k!$, and the definition of normalized polynomial form follows from the aforementioned.

Algorithm 2: Filter procedure

Input: The polynomial form of $f(t)$, coefficient precision $p_2 = n^{50}$ digits and r_{sp}
Output: The polynomial form of $o(t)$ with precision p_2 digits
sp is simply s in this pseudocode with r_{sp} written as r_s;
All polynomial forms assumed to be of degree n_d, except for step $s = 1$ which is of degree $n_{d,1}$;
$*$ refers to multiplication, with the product result truncated to precision p_2 digits;
Let $p(t) = e^{-t}$, and $p[i]$ represents normalized polynomial coefficient to degree $n_{d,1}$;
Let temporary storage be $j[]$ (vector), initialized with the normalized polynomial form of $f(t)$;
Let temporary storage be $j_p[]$, initialized with 0's;
Let temporary storage be q and w, initialized with 0;
Let m be initialized with $n_{d,1}$;
$d, i \leftarrow 0$; $s \leftarrow 1$;
repeat
  if $s \neq 1$ **then**
    $m \leftarrow n_d$;
  repeat
    repeat
      if $i + d + 1 \leq m$ **then**
        $j_p[i + d + 1] \leftarrow j_p[i + d + 1] + j[d] * p[i]$;
      $i \leftarrow i + 1$;
    until $i > m$;
    $i \leftarrow 0$; $d \leftarrow d + 1$;
  until $d > m$;
  $d \leftarrow 0$;
  repeat
    $w \leftarrow w + j_p[i] * (r_s)^i / i!$;
    $q \leftarrow q + (r_s)^i parity(i)/i!$;
    $i \leftarrow i + 1$;
  until $i > m$;
  $\mu \leftarrow -w/q$; $q, w, i \leftarrow 0$;
  repeat
    $j_p[i] \leftarrow j_p[i] + \mu * parity(i)$;
    $i \leftarrow i + 1$;
  until $i > m$;
  $i, d \leftarrow 0$; $s \leftarrow s + 1$;
  $j[] \leftarrow j_p[]$; $j_p[] \leftarrow 0$;
  $s \leftarrow s + 1$;
until $s > n_d + 3$;

2.6 Pseudocode for pseudo-steps

The algorithm is in Algorithm 3

2.7 Pseudocode for extraction of n_h

The algorithm is provided in Algorithm 4. $o(t)$ refers to $o(t)$ in Algorithm 2, $\phi'_{0,1}$ and $\phi'_{1,1}$ used as in Algorithm 3.

3 Analog approach

The analog approach is discussed completely separately from the algorithmic (digital) approach. Thus, both sections can be read in standalone ways. The analog approach is mentioned, because as seen in the algorithmic approach, analog signal processing was the foundation, instead of digital signal process-

Algorithm 3: pseudo-step procedure

Input: None. Assumed input e^{-t}.
Output: $\phi'_{0,1}$ and $\phi'_{1,1}$
sp is simply s in this pseudocode with r_{sp} written as r_s;
All polynomial forms assumed to be of degree n_d;
$*$ refers to multiplication, with the product result truncated to precision p_2 digits;
Let $p(t) = e^{-t}$, and $p[i]$ represents normalized polynomial coefficient to degree n_d;
Let temporary storage be $j[]$ (vector), initialized with the normalized polynomial form of e^{-t};
Let temporary storage be $j_p[]$, initialized with 0's;
Let temporary storage be q, w, initialized with 0;
Let m be initialized with n_d;
$d, i \leftarrow 0$; $s \leftarrow 2$;
repeat
  repeat
    repeat
      if $i + d + 1 \leq m$ **then**
        $j_p[i + d + 1] \leftarrow j_p[i + d + 1] + j[d] * p[i]$;
      $i \leftarrow i + 1$;
    until $i > m$;
    $i \leftarrow 0$; $d \leftarrow d + 1$;
  until $d > m$;
  $d \leftarrow 0$;
  repeat
    $w \leftarrow w + j_p[i] * (r_s)^i / i!$;
    $q \leftarrow q + (r_s)^i parity(i)/i!$;
    $i \leftarrow i + 1$;
  until $i > m$;
  $\mu \leftarrow -w/q$; $q, w, i \leftarrow 0$;
  repeat
    $j_p[i] \leftarrow j_p[i] + \mu * parity(i)$;
    $i \leftarrow i + 1$;
  until $i > m$;
  $i, d \leftarrow 0$; $s \leftarrow s + 1$;
  $j[] \leftarrow j_p[]$; $j_p[] \leftarrow 0$;
  $s \leftarrow s + 1$;
until $s > n_d + 3$;
$\phi'_{0,1} \leftarrow j[0]$; $\phi'_{1,1} \leftarrow j[1]$;

ing - basically, polynomial (analog signal processing foundation) vs. samples (digital signal processing) for the form of inputs and outputs in the algorithmic approach. Thus the analog approach is more "natural" way of understanding, even though it has less relevance in reality and may not even be operative. Anyone solely interested in the digital computer algorithm for obtaining the number of hamiltonian paths in undirected graph $G = (V, E)$ will gain nothing from this section. Nevertheless because of the history behind the algorithmic approach, the analog approach is mentioned also.

Before going on, in practice and in theory there is no good frequency multiplier except for a signal of single frequency sinusoid. Thus, the use of frequency multiplier is a conceptual one. An alternative circuit formulation, used somewhat identically for the algorithmic approach, is also shown and used.

In general, analog methods do suffer from several problems, and this should be in consideration when reading off the results in this section.

Graph $G = (V, E)$ is defined with a set V of vertices along with a set E of undirected edges connecting vertices. We will denote a walk by the following

Algorithm 4: n_h extraction procedure

Input: $\phi'_{0,1}$, $\phi'_{1,1}$, $o(t)$, $|V| = n$, r_μ, $r_{\mu+1}$
Output: n_h
Use system-of-two-equations-solver for Equation 19, with $o[0] = c_0$, $o[1] = c_1$ and given inputs;
Round the resulting n_h to the nearest integer;

formalism: $a-b-c$ where a, b, c are vertices and $-$ represents edges. Generally, a, b, c will be represented with positive integers.
$n = n_v$ is the cardinality of V, n_e is the cardinality of E. A n-walk is defined to be a walk with n vertices. This is the only class of walks we will have interests in this paper.
We will re-interpret Hamiltonian path existence problem using a $n \times n$ grid and others.

Definition 3.1. The grid contains n vertical columns. Each column contains n vertices, and the vertices in the same column are not connected by wires.

Definition 3.2. All vertices are numbered with positive integers greater than 1.

Definition 3.3. Each wire transmits a voltage signal $f(t)$. For our consideration, location does not matter, so all of our signals are solely function of time. These signals can be transformed into the Fourier transform frequency representation.

Definition 3.4. As part of lumped circuit assumption, we will assume that wires have no time delay. (ideal wire)

Definition 3.5. For each vertex x at column $a > 1$, if vertex y satisfies $(x, y) \in E$ or $(y, x) \in E$, vertex y frequency multiplier (or oscillator, in case of $a-1 = 1$) at column $a-1$ is connected by a wire to the sum operator at vertex x/column a.

Definition 3.6. As we allow self-loops, while $(x, x) \notin E$, vertex x at column $a-1$ is connected by a wire to the sum operator at vertex x/column a.

Definition 3.7. Each vertex x at column 1, the first column, only has an ideal oscillator, transmitting e^{ixt} to wires connected to the second column.

Definition 3.8. A sum operator just sums up the signals transmitted by wires.

Definition 3.9. Each sum operator at vertex x/column a is connected to a frequency multiplier at the same vertex/column, with frequency multiplication factor of x. Frequency multiplier transforms $e^{iw_1 t} + e^{iw_2 t} + ...$ into $e^{ixw_1 t} + e^{ixw_2 t} + ...$

Definition 3.10. At column n, after signals pass through frequency multipliers connected to sum operators, any wire incident from column n is connected to a final sum operator, which produces the final signal $y(t)$.

Thus it is clear that we need $n(n-1)+1$ sum operators (or adders, equivalently) and $n(n-1)$ frequency multipliers for the grid above. The number of wires are dependent on E, but the maximum number of wires required is $n^2(n-1)+n(n-1)+n$, where the last n comes the wires that connect column n to the last sum operator, and $n(n-1)$ comes from the wires that connect a single sum operator to a single frequency multipliers.
The output of the circuit grid defined above is $y(t)$, as mentioned above. Let $V = \{v_1, v_2, .., v_n\}$.

Definition 3.11. The final sum operator, which produces the signal $y(t)$ is connected to the ideal mixer M, which outputs the product of $y(t)$ with e^{-iut} where $u = v_1v_2v_3..v_n$. In Fourier transform, this is equivalent to converting $Y(\omega)$ with $Y(\omega+u)$, where $Y(\omega)$ is Fourier transform of $y(t)$. Let the output of M be $k(t)$.

From the above, it is clear that Ce^{iut} inside $y(t)$ represents hamiltonian paths, with C representing the number of hamiltonian paths. In $k(t)$, frequency 0 represents hamiltonian paths, as all frequencies are shifted left by u.
Because our chosen low-pass filter will be first-order, we will also pass $k(t)$ to a frequency multiplier that multiplies frequencies by ${v_n}^{4n}$ where v_n is the greatest-numbered vertex, to ensure that the frequencies other than zero frequency parts of $k(t)$ will be sufficiently high frequencies. (Multiplying zero by ${v_n}^{4n}$ is zero) For higher-order filters, like third-order filter, this additional frequency-multiplying process will not be needed. We will call the resulting signal $j(t)$.
As a side note, instead of having input tape in Turing machine, we have to re-wire $n \times n$ grid every time graph input changes. This $n \times n$ grid serves as an input to the system involving a low-pass filter.

3.1 Restriction on vertex indices

However, a close look will reveal that it is required for us to restrict on vertex indices. Hamiltonian u may be decomposed into a product of n numbers that are in V, and yet all these numbers may not be distinct, required for u to represent hamiltonian paths. One simple way to address this problem is by required all vertex indices to be prime numbers. For simplification, assume that $v_1 = 2$ and $v_n = p_n$ where p_k represents kth prime number with $p_1 = 2$. It is known that $p_n < n(\ln n + \ln\ln n)$, shown in Rosser (1941). Thus we only need to check non-exponential number of natural numbers to obtain n prime numbers to be used as indices for vertices.

3.2 Analog approach: low-pass filter

Now that we defined the final output $k(t)$, the question is how we process $k(t)$ to give us some information about the number of hamiltonian paths, or C. To do this, we pass it to a low-pass filter. But we cannot simply assume an ideal low-pass filter, represented by $H(\omega) = rect(\omega)$, where $rect(\omega) = 1$ for $-0.5 < \omega < 0.5$ and $rect(\omega) = 0$ otherwise, because there is no such an ideal

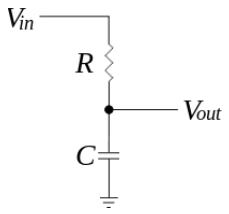


Figure 3: A first-order RC low-pass filter

filter even to the approximate level.
Thus we will choose a simple physical first-order RC low-pass filter, described in figure 3. (The figure is in public domain.)
By Kirchhoff's Voltage Law, the low-pass filter in figure 1 has the ODE of:

$$\frac{dV_{out}}{dt} + \frac{V_{out}}{\tau} = \frac{V_{in}}{\tau}$$

where $\tau = RC$. As this ODE is linear, to figure out the behavior of this low-pass filter, we first consider $V_in = De^{i\omega t}$, where ω is some arbitrary frequency.
Using initial capacitor voltage condition at the starting time $t = 0$ as $V_{out:t=0} = 0$,

$$V_{out} = \frac{D}{1 + i\omega\tau}\left[e^{i\omega t} - e^{-t/\tau}\right]$$

Assume that $v_n > n + 1$. Also for calculation convenience, assume that $\tau = RC = 1$. In steady state $t = \infty$, because every ω of $j(t)$ except zero is greater than/equal to ${v_n}^{4n}$, and the total number of walks in G with n total vertices can only have maximum of n^n n-walks, $j(\infty)$'s value mostly comes from the hamiltonian/zero-frequency part. Other frequency parts only contribute less than $1/n^{3n}$ in magnitude. Thus at time ∞, the number of hamiltonian paths is discovered from the magnitude of $j(\infty)$, $|j(\infty)|$. However, calculations must be done on finite time, so the steady-state case only forms a background for our discussions, not the main part.
Note that in ordinary signal processing, keeping phase errors small is very important, but for the use of signal processing tools to analyze hamiltonian paths, phase errors are not of any concern.

3.3 Time Complexity of the Circuit

But moving to the finite time is simple: figure out the time when $e^{-t/\tau}$ decays to $1/n^{4n}$. Then high frequency parts only contribute a negligible value to $j(t)$. Now since $\tau = 1$ assumption is made, set equality $e^{-t_c} = 1/n^{4n}$. Taking the natural log to each side, $t_c = 4n \ln n < 4n^2$. Thus, the critical time, which is when the exponential decaying factor decays to $1/n^{4n}$, increases approximately linearly as the size of input n increases.
After this critical value, the value of $|j(t)|$ can simply be sampled by a digital

computer to get the number of hamiltonian paths. Note that theoretically only one sample is required to measure the number of hamiltonian paths. This is because frequency 0 does not have any oscillating part, and thus will have constant offset relative to 0.
Thus time complexity of the circuit to solve the number of hamiltonian paths is $O(n \log n)$, which is smaller than $O(n^2)$.

3.4 Size and Time Complexity

The above demonstrates that the number of needed components and needed time does not grow exponentially as the input graph size increase. All the values used in the circuit does not require exponentially-growing number of digits in a digital computer, as the graph size increases.

3.5 Alternative circuit formation

In this section, I will describe another way of building a circuit that represents a graph. This method eliminates the use of frequency multipliers, and replaces them with ordinary multipliers.
Start with the original idea that each vertex x at column 1 transmits e^{ixt} to the wires x at column 1 are connected to. All wires going to vertex y at column $i > 1$ are first met with a sum operator, but now followed by an ordinary multiplier of $sum \times e^{iyt}$. The method will be explained in detail below.
Definition 3.1, 3.2, 3.3, 3.4 will be used as before. Section 3.1 no longer applies and is replaced with the restrictions imposed by the following definitions in this subsection:

Definition 3.12 (The set V of vertex numbers)**.** The set V is defined as $V = \{n, n^2, ..., n^n\}$, which represents the set of vertex numbers (or equivalently vertex indices), with $|V| = n$, the number of vertices.

Definition 3.13 (n-walk)**.** A n-walk $\xi = (\xi_1, \xi_2, .., \xi_n)$ with $\xi_i \in V$ and $(\xi_i, \xi_{i+1}) \in E$ or $\xi_i = \xi_{i+1}$, a list, is a walk that has n vertices. A n-walk may contain self-loops or loops. One may consider a n-walk as a list of n vertex numbers that may contain one vertex number more than once.

Definition 3.14 (Permutation of a list)**.** A permutation of a list is a re-ordering of list elements of ξ.

Definition 3.15 (Uniqueness of n-walk frequency)**.** Let a n-walk ξ be $\xi = (\xi_1, \xi_2, .., \xi_n)$, which is a list. Let $\omega = \sum_{i=1}^{n} \xi_i$. ω is a unique n-walk frequency of G if it can only be the sum of some permutations of one list.

Lemma 3.1. *For $V = \{n, n^2, ..., n^n\}$, there cannot exist a n-walk frequency such that it is not unique.*

Proof. The proof is simply the basis representation theorem, except that the case where n vertex numbers that are same are in the list. In such a case,

$\omega = n \cdot n^i$. But then $\omega = n^{i+1} = 1 \cdot n^{i+1}$, and $\xi = (n^{i+1})$ is the only possible alternative representation of ω. But the alternative list only has one vertex. Thus, there cannot exist a n-walk frequency that is not unique. □

Definition 3.5 needs to change as follows:

Definition 3.16. For each vertex x at column $a > 2$, if vertex y satisfies $(x, y) \in E$ or $(y, x) \in E$, vertex y mixer at column $a - 1$, which multiplies e^{iyt} to a signal it receives, is connected by a wire to the sum operator at vertex x/column a. In case of each vertex x at column $a = 2$, if vertex y satisfies $(x, y) \in E$ or $(y, x) \in E$, vertex y oscillator (output of e^{iyt}) at column 1 is connected by a wire to the sum operator at vertex x/column 2.

Definition 3.6, 3.7 and 3.8 are kept. Definition 3.9 and 3.10 change to the following:

Definition 3.17. Each sum operator at vertex x/column $a \geq 2$ is connected to a mixer at the same column and the same vertex, which shifts frequency by x. A mixer, with shift factor of x, transforms $e^{iw_1 t} + e^{iw_2 t} + ...$ into $e^{i(w_1+x)t} + e^{i(w_2+x)t} + ...$, because it multiplies e^{ixt} to the signal it receives.

Definition 3.18. At column n, after signals pass through mixers connected to sum operators, any wire incident from column n is connected to a final sum operator instead, which produces the final signal $y(t)$.

Complexity remains the same: one needs $n(n-1)+1$ sum operators and $n(n-1)$ mixers/multipliers. (multipliers here are not frequency multipliers, but ordinary signal multipliers) The number of wires required remains the same.
Definition 3.11 changes to the following:

Definition 3.19. The final sum operator, which produces the signal $y(t)$ is connected to the ideal mixer M, which outputs the product of $y(t)$ with e^{-iut} where $u = v_1 + v_2 + v_3 + .. + v_n$, with $v_i \in V$. In Fourier transform, this is equivalent to converting $Y(\omega)$ with $Y(\omega + u)$, where $Y(\omega)$ is Fourier transform of $y(t)$. Let the output of M be $k(t)$.

Now $k(t)$ has zero frequency as its hamiltonian path frequency, as in the original formulation.
One may choose to add frequency multiplier after the final mixer M so that a simple first-order low-pass filter can be used. However, one may instead choose to increase the difference between each vertex number, such as $V = \{n, n^n, n^{2n}, ..., n^{n^2}\}$. This way, one does not have to add an extra frequency multiplier, which is likely to diverge from its ideal behavior, as I will discuss.

3.6 Real deviations: perils of high frequency and frequency multipliers

While the system described above is a physical system, not just a logical system, it is nevertheless still an ideal physical system. Oscillators are not perfect

oscillators, resistors and capacitors are not ideal ones, wires have impedance. Thermal effects may change system properties.
But the most fundamental problem is the fact that the systems above are based on lumped-circuit analysis. Lumped circuit analysis works for low frequencies, because the length of wires can be made short enough to satisfy lumped-circuit assumptions. But one cannot shorten wires forever, and this makes lumped-circuit analysis to break for high frequencies. No longer discussion of lumped capacitors, resistors and inductors becomes a simple one.
Many problems, whether small or not, require more details and are left out here. Future papers will address these issues.

4 Conclusions

For the algorithmic/digital approach, the conclusions are provided in the separate subsection. Note, or recall, again that digital and analog approaches were discussed separately and thus they can be read separately without affecting understanding. Thus, if the interest of the reader is solely on the algorithmic approach, one may skip reading the entire analog approach section and this conclusion. For the analog approach, the conclusion is that while ideal analog computing models - one like GPAC - provide a useful tool, they nevertheless fail for the problem we would like to address for all realistic implementations of analog computers. The question thus is how close can a real analog computer be to ideal computing models (this involves sciences, like physics, chemistry and biology) and under current realistic limitations, how improvements can be made. Some possible improvements have been provided in this paper, but rest are still left to future papers. Furthermore, while the digital approach is inspired by analog signal processing methods, using the digital approach may in the end be much better than the analog approach.

4.1 An alternative model for analog computations

As we have seen in the paper, analog signal processing is the foundation and is easily converted to an analog computer circuit. However, because of error build-up in analog computers, digital computing is usually preferred for large n (and in practice, no one uses an analog computer).
So far, the GPAC ideal analog computer model was used for the analog approach. However, one may also think of distributed analog computation models [5]. The idea is to provide a compromise between analog and digital approaches. The GPAC model assumes infinite precision digit, which is not the case in practice. We may try to do get the best analog precision for wire precisions and components, but one may instead opt to distribute some of the digits. For example, if one requires 50-digit precision, then one may divide this precision into 10 5-digit-precision wires/components, with components interacting with each other (and processors are analog, instead of digital). As [5] shows, this provides improvements in analog signal-to-noise ratio. This, however, does not

completely eliminate significant noise accumulation over "stages" (to refer to the cases in this paper, "depths" or "levels" in a grid) as number of stages passed by increases. This makes us think of the bybrid approach in [5], which involves digital restoration and use of A/D/A.
Here, one sees some common grounds with what was done in the algorithmic approach, even though analogy will not be perfect. I will leave more detailed arguments to future papers, and conclude this paper.

References

[1] Olivier Bournez, Manuel L. Campagnolo, and Daniel S. Graca. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, 2007.

[2] Hendrik Willem Lenstra, Arjen Klaas Lenstra, and Laszlo Lovasz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[3] Katsuhiko Ogata. *Modern Control Engineering*. Pearson, 5 edition, 2009.

[4] Alan V. Oppenheim and Alan S. Willsky. *Signals and Systems*. Pearson, 2 edition, 1996.

[5] Rahul Sarpeshkar. Analog versus digital: Extrapolating from electronics to neurobiology. *Neural Computation*, 10(7):1601–1638, 2014.

[6] Claude E. Shannon. Mathematical theory of the differential analyzer. *Studies in Applied Mathematics*, 20(1-4):337–354, 1941.